Extensão Protocolar para Comunicação Fiável em Ambientes IoT

Protocolar Extension for Reliable Communications in IoT Environments

João F. Silva, Carlos S. Lopes
Departamento de Informática
Universidade do Minho
4750 059 Braga, Portugal
e-mail: {a72023,pg31544}@alunos.uminho.pt

Pedro Sousa, Paulo Carvalho
Centro Algoritmi, Departamento de Informática
Universidade do Minho
Braga, Portugal
e-mail:{pns,pmc}@di.uminho.pt

Resumo — **Acompanhando o desenvolvimento atual na área das Tecnologias de Informação, e sobretudo a nível das Redes de Computadores, é notório o avanço científico e tecnológico da área designada por *Internet of Things* (IoT). Neste contexto, este artigo centra-se na proposta e desenvolvimento de uma extensão protocolar de comunicação vocacionada para ambientes IoT. A solução proposta foi desenhada para proporcionar capacidade de comunicação *multi-hop* fiável para dispositivos IoT de uso generalizado e plataforma aberta (e.g., Arduino). A API implementada fornece um conjunto modular de primitivas de serviço que permitirá suportar o desenvolvimento de aplicações IoT com requisitos diferenciados de fiabilidade e abrangência.**

Palavras Chave - Suporte a aplicações IoT; Internet das Coisas; Redes de Sensores sem Fios; Protocolos de Comunicação Fiáveis.

Abstract — **The current developments in the Information Technology area, and particularly in Computer Networks related fields, have sustained a considerable scientific and technological progress of the area usually denominated as the Internet of Things (IoT). In this context, this article focuses on the design, implementation and test of a communication protocol extension oriented to IoT environments. The proposed solution is designed to provide reliable multi-hop communication capabilities for commonly used and open platform IoT devices (e.g. Arduino). The implemented API provides a modular set of service primitives that will allow the successful development of IoT applications with differentiated reliability and coverage requirements.**

Keywords - Support to IoT Applications; Internet of Things; Wireless Sensor Networks; Reliable Communication Protocols.

I. INTRODUÇÃO

O conceito de IoT, embora recente, está cada vez mais presente no nosso dia-a-dia, desde dispositivos eletrónicos de uso generalizado, a automóveis, ou cidades inteligentes (*smart cities*). A sua proliferação está associada à inovação em diversos campos muito importantes, tais como *Wireless Sensor Networks* (WSNs) [1], inteligência artificial ou nanotecnologia.

Neste artigo, a contribuição está relacionada com a inovação e o desenvolvimento das WSNs, promovendo o aumento do seu uso através da melhoraria da versatilidade e desempenho protocolar. Em particular, o presente artigo propõe uma extensão protocolar que permite comunicação fiável em ambientes WSN *single-hop* e *multi-hop*. Seguindo a tendência de outros projetos semelhantes desenvolvidos pela comunidade e tendo em consideração a necessidade de manter um baixo custo, neste estudo serão utilizados como dispositos IoT Arduinos conjugados com transmissores sem fios nRF24L01+ [2].

Estas plataformas, como referido anteriormente, conferem grande suporte devido à sua popularidade, sendo que existem já algumas bibliotecas *Open Source* disponíveis para facilitar o desenvolvimento e a integração deste tipo de sistemas no mundo real. Sendo que estas bibliotecas são desenvolvidas muitas vezes em contextos específicos e abrangência limitada, são portadoras de algumas limitações, tais como, reduzidas funcionalidades ou problemas de desempenho. Deste modo, torna-se relevante a resolução ou atenuação destas limitações, de forma a tornar os sistemas mais capazes e usáveis em diferentes contextos aplicacionais. Posteriormente, neste artigo, serão detalhadas as funcionalidades já oferecidas pelo conjunto de bibliotecas acima mencionadas.

Devido ao curto alcance dos transmissores nRF24L01+, caso a WSN em causa esteja dispersa por uma área muito alargada torna-se muito difícil garantir a comunicação entre todos os nós em *single-hop*. Surge, então, a necessidade de equipar a WSN com recursos necessários para ultrapassar esta dificuldade. Este problema é parcialmente resolvido pelas bibliotecas já existentes. No entanto, estas não permitem que as comunicações em *multi-hop* sejam confirmadas fim-a-fim, i.e. entre qualquer nó de origem e destino. Estes requisitos de fiabilidade serão especialmente importantes quando se está na presença de aplicações que lidam com dados mais críticos, tais como aplicações na área da emergência médica ou ambiental, entre outros variados exemplos.

Assim, o propósito deste trabalho consiste em oferecer uma solução flexível que permita a comunicação fiável em redes WSN *multi-hop*. Esta funcionalidade será fornecida por uma camada de software que utiliza por base, e extende bibliotecas

já existentes. Assim, a aplicações que utilizam um cenário semelhante, é oferecida a garantia de fiabilidade nas comunicações entre todos os nós que constituem a WSN. Esta camada protocolar, similarmente ao que acontece na pilha TCP/IP [3], vai permitir que sobre ela sejam facilmente desenvolvidas aplicações IoT que necessitem satisfazer este tipo de requisito.

Este documento está organizado como se segue: a introdução e um breve estudo do transmissor nRF24L01 é realizado na Secção II, enquanto a plataforma de desenvolvimento utilizada é descrita na Secção III. A Secção IV inclui a modelação da extensão protocolar proposta descrita em *Unified Modeling Language* (UML), os detalhes da *Application Programming Interface* (API) correspondente, bem como a prova de conceito através de duas aplicações de teste que utilizam a API desenvolvida. Por último, as conclusões são incluídas na Secção V.

II. *Enhanced ShockBurst™* (ESB) e NRF24L01+

A. *Enhanced ShockBurst™*

O ESB [4] é um protocolo de nível 2 oferecido com o chip nRF24L01+ que disponibiliza um nível básico de comunicação para uma topologia de rede em estrela [5]. Este protocolo suporta envio bidirecional de pacotes de dados com confirmação e retransmissão em *single-hop*. Os dados a enviar e a receber passam por um *buffer* FIFO antes de serem transmitidos.

O protocolo ESB está incorporado em todo o hardware produzido pela *Nordic Semiconductor* da série *nRF24Lxx*.

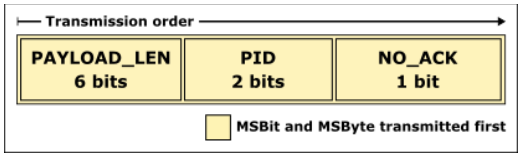

Figura 1. PCF [6]

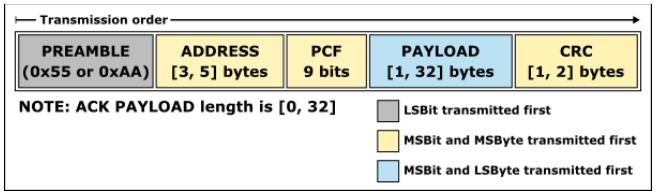

Figura 2. PDU ESB [6]

Todos os dispositivos que usam o ESB deverão estar associados a um endereço com um comprimento entre 3 e 5 bytes, utilizado para identificar as diferentes estações da rede.

De forma a distinguir as diferentes tramas, suportar um *payload* variável e possibilitar que os dados *single-hop* não necessitem de ser confirmados, encontra-se disponível o campo *Packet Control Field* (PFC), ver Figura 1.

A carga útil do *Protocol Data Unit* (PDU) a este nível é no máximo de 32 bytes, podendo a mesma variar, como referido anteriormente. De modo a sincronizar os transmissores envolvidos na comunicação, é enviado um preâmbulo no início de cada trama, podendo este ser *0x55* (*01010101*) ou *0xAA* (*10101010*) dependendo do 1º bit do endereço do nodo destino. Para permitir controlo de erros existe a opção de ativar 8 ou 16 bits para a verificação com *Cyclic Redundancy Check* (CRC) [7]. Estes campos encontram-se representados na Figura 2.

B. *O transmissor nRF24L01+*

O nRF24L01+ é um transmissor rádio que funciona na gama de frequências entre 2.4GHz e 2.525GHz, existindo assim a possibilidade de seleção entre 125 canais (RF_CH) para comunicação distintos, cujas frequências se encontram separadas por 1MHz. A frequência de transmissão alocada para cada canal pode ser obtida através da Equação 1 [2].

$$FCh[MHz] = 2400 + RFCh \quad (1)$$

A implementação do protocolo ESB encontra-se embebida no próprio *chip*, pelo que estão disponíveis todas as características associadas ao ESB, referidas na Secção II-A.

Este transmissor oferece a possibilidade de configurar diferentes velocidades de transmissão, sendo o valor máximo permitido de 2Mbps. Esta velocidade de transmissão, aliada aos modos de poupança de energia disponíveis, torna este dispositivo muito atrativo para sistemas onde as questões energéticas são uma preocupação [2].

A comunicação para troca de comandos e dados entre o transmissor e o seu *host* é efetuada através de uma interface *Serial Peripheral Interface* (SPI). O SPI é um protocolo de comunicação com fios muito comum nas comunicações entre micro-controladores. Neste protocolo existem dois tipos de intervenientes: o dispositivo *master*, que controla todo o protocolo de comunicação, e *N* dispositivos *slaves* que comunicam com o *master*, assim que o mesmo entender. Trata-se de um protocolo síncrono, onde existe uma linha dedicada para sincronização de relógios entre os dispositivos. Funciona em *full-duplex*, ou seja, existem duas linhas separadas para a receção e envio de dados. Existe, ainda, um canal dedicado ao suporte da utilização de vários dispositivos no mesmo canal de dados, no qual o dispositivo *master* anuncia qual o *slave* que possui a alocação desse canal naquele intervalo de tempo [8].

Por fim, acerca do transmissor utilizado, é importante referir que o nível físico da comunicação neste *chip* é garantido por uma modulação denominada *Gaussin Frequency Shift Keying* (GFSK).

III. Plataforma de Desenvolvimento

Como mencionado anteriormente, nesta secção serão expostas, detalhadamente, todas as bibliotecas *OpenSource* utilizadas para interagir com o transmissor *nRF24L01+*. Desde a biblioteca mais simples, descrita na Secção III-A, que apenas mascara os comandos que o transmissor aceita, até uma API mais intuitiva de usar pelo programador. Em cima desta biblioteca foram desenvolvidos vários níveis de rede lógicos, onde encontramos a implementação de uma WSN utilizando os transmissores *nRF24L01+*, apresentada na Secção III-B. Por fim, na Secção III-C, apresenta-se uma biblioteca que simplifica a uso da WSN em questão com a nova API (ver Secção III-B) e por isso é utilizada como base de desenvolvimento ao trabalho a efetuar.

A. *Biblioteca RF24*

A biblioteca RF24 é uma biblioteca implementada em C++ que tem como principal objetivo garantir uma interação mais fácil com o transmissor nRF24L01+ (ver Secção II-B). Ao utilizar esta biblioteca como base de desenvolvimento, todo o *software* desenvolvido sobre ela tem como vantagem associada o facto do programador não se ter de preocupar em fazer a interação com o transmissor via SPI, onde todos os comandos são especificados no manual do transmissor [2].

Assim, ao utilizá-la, consegue-se interagir com o transmissor de uma forma mais amigável para o programador onde é oferecida uma API que possibilita, de uma forma simples e eficaz, a interação com o transmissor.

Entre todos os métodos disponibilizados pela biblioteca RF24 destacam-se os que oferecem as seguintes funcionalidades:

- definir o canal de comunicação;
- ativar e desativar ACK;
- definir o CRC para controlo de erros;
- parametrizar velocidade e potência de operação;
- ligar e desligar o transmissor;
- enviar e receber qualquer tipo de dados;
- deferir endereços dos *pipes* utilizados na comunicação.

É de salientar que ao utilizar esta biblioteca nenhum cabeçalho adicional é utilizado, pelo que, o *payload* máximo por PDU continua a ser de 32 bytes.

Toda a documentação, bem como exemplos associados a esta biblioteca, encontram-se disponíveis em [9].

B. *Biblioteca RF24Network*

A utilização desta biblioteca permite que as mensagens enviadas entre os nodos tenham a possibilidade de efetuar mais que um salto até chegar ao destino. Deste modo, fica resolvido o problema da comunicação entre nodos que se encontram a grandes distâncias, uma vez que esta biblioteca permite o envio de mensagens com recurso a *multi-hop*. Para dar suporte às novas funcionalidades, foi introduzido nestas bibliotecas um cabeçalho adicional, semelhante ao que se encontra nos diferentes níveis da pilha TCP/IP. Neste cabeçalho encontram-se vários campos disponíveis, tais como:

- endereços da fonte e destino da mensagem;
- numeração única;
- tipo associado ao conteúdo da mensagem;
- outros campos de controlo

Ao introduzir este novo nível de abstração, foi possível esconder o endereçamento com *pipes*, mencionado na Secção III-A, e passar a utilizar endereços mais fáceis de associar aos nodos. Porém, este tipo de endereçamento ainda apresenta algumas limitações sintáticas e semânticas.

À semelhança do que acontece nas tramas *Ethernet*, em que o campo *Type* dá a conhecer o tipo do *Payload* encapsulado [3], tem-se também, a este nível, alguma informação acerca do *Payload* contido em cada PDU. Contudo, nesta implementação, apenas é possível diferenciar 128 tipos distintos de conteúdo encapsulado.

Com a introdução deste cabeçalho observa-se que o *payload* do PDU é reduzido em 10 bytes, passando assim a ter um valor máximo de 22 bytes. Esta informação encontra-se representada na Figura 3.

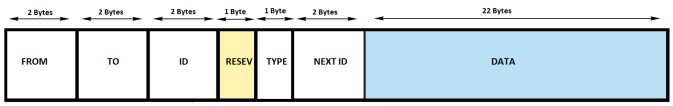

Figura 3. PDU RF24Network

Toda a documentação, bem como exemplos associados a esta biblioteca encontram-se disponíveis em [10].

C. *Biblioteca RF24Mesh*

Como foi abordado na Secção III-B, as funcionalidades *multi-hop* já são suportadas pela biblioteca enunciada. Apesar de já existir um suporte de base necessário para o desenvolvimento do que é proposto fazer, existem ainda certas lacunas e dificuldades ao utilizar a biblioteca RF24Network. Estes obstáculos, já enunciados na secção anterior, foram suavizados com a utilização da nova biblioteca RF24Mesh que disponibiliza uma atribuição automática de endereços RF24Network. Para possibilitar esta tarefa, o nodo principal (Nodo 0) deve suportar um serviço semelhante ao serviço DHCP, atribuindo os endereços RF24Network com base numa numeração decimal identificadora de cada nodo (até um limite de 255).

Associadas a esta finalidade, estão também disponíveis as funcionalidades de guardar e recarregar tabelas DHCP, libertar um endereço atribuído e, por fim, anunciar a disponibilidade para ser um *Super nó*, ou seja, permitir, ou não, que outros nodos descubram e se liguem ao nó atual.

É também de salientar que a introdução desta nova camada de *software* em nada altera a estrutura do PDU utilizado, não sofrendo assim nenhuma alteração face ao apresentado na Figura 3. No entanto, com esta nova camada, foi introduzido um *overhead* no que diz respeito ao número total de mensagens trocadas entre os nodos, bem como, à complexidade acrescida do algoritmo que corre agora em cada nodo do sistema devido à necessidade de guardar tabelas, onde são associados os endereços RF24Network e os endereços decimais agora utilizados.

Toda a documentação, bem como exemplos associados a esta biblioteca, encontram-se disponíveis em [11].

IV. Desenvolvimento da Biblioteca Proposta

Tendo como ponto de partida tudo o que foi apresentado na Secção III, será agora exposto o trabalho efetuado para dotar o sistema das funcionalidades de comunicações confirmadas em *multi-hop*. Inicialmente, na Secção IV-A, será apresentada a estrutura do novo PDU. Posteriormente, na Secção IV-B, são

apresentadas as máquinas de estado nas quais todo o protocolo de comunicação desenvolvido se baseia. Em forma de conclusão do desenvolvimento, e tendo em conta tudo que já foi referido, na Secção IV-C, são expostas as funcionalidades a que esta nova API deve ser capaz de responder. Para terminar, na Secção IV-D, são apresentados alguns exemplos práticos onde se faz uso da API desenvolvida.

A. *Introdução do novo PDUs*

Com a introdução deste novo nível protocolar, torna-se necessária a restruturação dos PDUs disponibilizados. Esta restruturação é motivada por vários requisitos, entre os quais:

- numeração dos pacotes a este nível;
- necessidade de menos tipos disponíveis;
- introdução de uma versão do protocolo;
- indicação dos tamanho de dados válidos.

De forma a criar o menor *overhead* protocolar possível, parte destes campos aproveitam o campo Type já disponível, presente na Figura 3, passando este campo a ser estruturado da forma apresentada na Figura 4 (esq.). Os campos relativos à versão e à numeração dos pacotes foram introduzidos no primeiro Byte do campo de dados do PDU original. Esta reorganização encontra-se apresentada na Figura 4 (dir.).

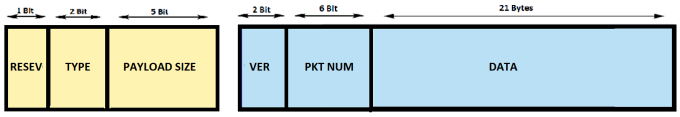

Figura 4. Restruturação do PDU RF24Network: Campo *Type* (esq.); Campo *Data* (dir.).

Assim, esta nova reconfiguração do PDU oferece:

- até 4 versões do protocolo distintas;
- 4 tipos de mensagens distintos;
- distinção do tamanho de mensagens até 32 Bytes;
- numeração de pacotes em módulo 64.

Com a reestruturação efetuada, tem-se agora disponível um campo de dados de 21 Bytes dos 32 Bytes iniciais oferecidos pelo ESB (ver Secção II-A).

B. *Diagramas de Transição de Estado do Protocolo*

Um dos grandes problemas encontrados por quem desenvolve aplicações em plataformas *single-thread* está associado à impossibilidade de executar duas partes de código em simultâneo. Sendo a plataforma de IoT *single-thread*, e existindo também a eventual necessidade de enviar e receber dados em simultâneo, foi implementada uma solução que, apesar de continuar a ser *single-thread*, tem como objetivo diminuir o atraso na transição entre os estados de receção e de envio de dados, evitando também o possível descarte de mensagens.

Nos mais variados protocolos de comunicação existentes, o comportamento protocolar é modelado por diagramas de transição de estados. Estes diagramas têm como principal objetivo prever e regulamentar todas as interações possíveis, durante a sua operação, de uma forma eficaz.

Nas Figuras 5 e 6 são apresentados os modelos que traduzem a receção e o envio de dados, e que suportam todo o protocolo a ser desenvolvido. Note-se que duas máquinas de estado desenvolvidas se encontrarem mutuamente interligadas por forma a contornar as restrições enunciadas anteriormente.

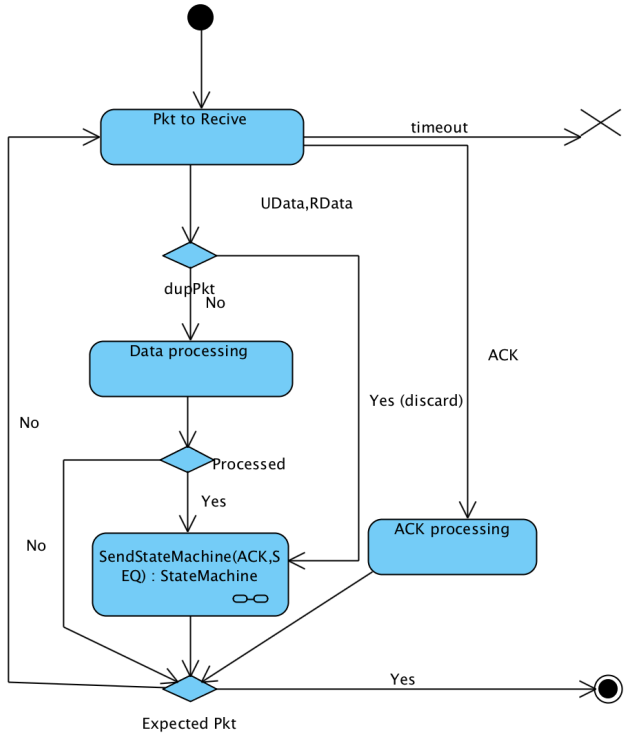

Figura 5. Máquina de estado do processo de receção de pacotes.

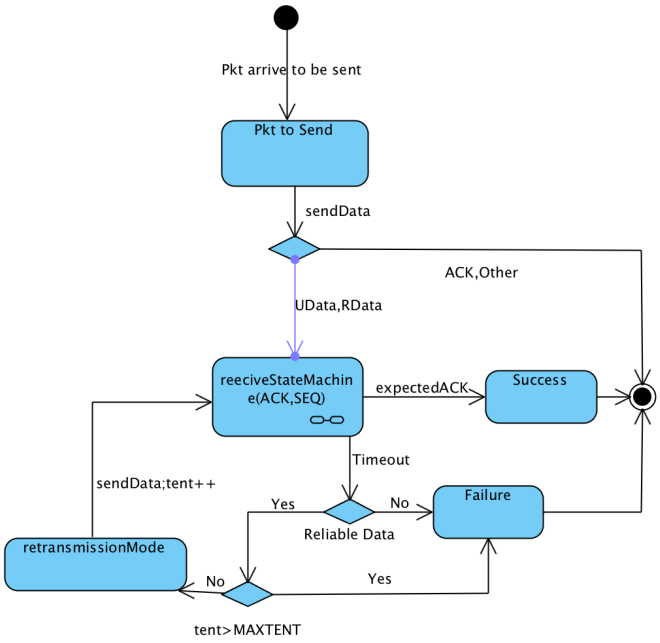

Figura 6. Máquina de estado do processo de envio de pacotes.

Por outro lado, no tratamento de sistemas distribuídos em comunicações, existem problemas comuns, tais como, mensagens que são perdidas ou que sofrem grandes atrasos. Isto obriga a que a receção de pacotes tenha em consideração estes casos. Para isso, foi introduzida a capacidade de processar pacotes que não são os esperados, bem como a introdução de um tempo limite (*timeout*) na receção de pacotes.

Um exemplo prático desta situação é originado quando um *Sistema A* envia uma mensagem para o *Sistema B*. De seguida, o *Sistema A* fica à espera de uma confirmação proveniente do *Sistema B*. Esta confirmação pode nunca chegar. Contudo, durante o tempo em que o *Sistema A* se encontra a aguardar esta confirmação pode chegar uma nova mensagem do *Sistema C* destinada ao *Sistema A*, pelo que se torna imperativo processá-la e enviar a confirmação da sua receção, voltando depois o *Sistema A* ao estado de espera da confirmação proveniente do *Sistema B*.

C. API Desenvolvida

Para além da funcionalidade de envio de dados confirmados e não confirmados, existe ainda a necessidade de fornecer às aplicações utilizadoras desta camada aplicacional uma total transparência face às bibliotecas referidas na Secção III. Deste modo, é necessário fornecer métodos/funções que possibilitem aceder a essas funcionalidades para que não seja necessário invocar as diferentes bibliotecas de base.

A este nível, devido a questões enunciadas na Secção IV-B, torna-se também impreterível a existência de filas de espera capazes de guardar temporariamente os pacotes aplicacionais recebidos quando não são esperados, tal como ilustrado no exemplo descrito na Secção anterior. Assim sendo, são também necessárias formas capazes de lidar com estas filas eficientemente.

Tendo em conta os aspetos previamente referidos, a API desenvolvida contempla os seguintes métodos:

- *update*: este método deve ser invocado frequentemente, sendo responsável por manter o sistema ativo;
- *receiveData*: método que permite a receção de mensagens;
- *sendRData*: método que possibilita o envio de mensagens fiáveis entre nodos;
- *sendUData*: método que proporciona o envio de mensagens não fiáveis entre nodos, semelhante ao já oferecido;
- *checkPendingReception*: método que proporciona a consulta de receções pendentes em fila de espera;
- *checkNextReception*: método que faculta a obtenção de mensagens pendentes na fila de espera.

Semelhante ao que se passa com as bibliotecas enunciadas na Secção III, e pelo facto do software ter sido desenvolvido a pensar em dispositivos *single-thread*, o método *update* deve ser chamado com frequência de forma a manter todo o sistema operacional. Este método foi desenvolvido de forma a ultrapassar as limitações dos sistemas *single-thread*.

Devido à existência de uma fila de espera para receção, quando o método *receiveData* recebe um novo PDU será feito o *enqueue* desse PDU. Para lidar com estas filas de espera são oferecidos os métodos *checkPendingReception*, que indica se existe algum PDU presente na fila à espera de ser processado, e o *checkNextReception,* que retira o próximo PDU da fila e o entrega à aplicação. Para realizar o envio de dados, estão disponíveis os métodos *sendRData*, que permite o envio de dados que necessitam garantia de entrega, e *sendUData*, que não garante a entrega dos dados.

D. Prova de conceito e Testes Preliminares

Como prova de conceito inicial, procurou-se demonstrar as funcionalidades pretendidas e oferecidas pela extensão protocolar desenvolvida através da realização de alguns testes preliminares. Semelhante ao que acontece nas bibliotecas utilizadas (ver Secção III), onde são disponibilizados exemplos de utilização, serão apresentados dois exemplos que fazem uso da camada protocolar e software desenvolvidos.

O primeiro teste tem como principal objetivo demonstrar as novas funcionalidades protocolares oferecidas. Trata-se de um teste inicial básico não diretamente relacionado com IoT. Neste teste foi possível observar os nodos secundários a enviarem mensagens de texto simples ("Hello World") para o nodo principal. Ao receber estas mensagens, o nodo principal faz o *output* das mesmas imprimindo-as na sua consola. Neste teste, todas as mensagens são enviadas com requisitos de fiabilidade. Perante a ocorrência de perda de mensagens, observou-se a sua retransmissão até confirmação da entrega das mesmas. Este teste foi delineado para funcionamento em *single-hop*, sendo facilmente estendido para múltiplos nodos da WSN.

Por outro lado, o segundo teste utiliza a nova biblioteca como base de suporte para uma aplicação mais realista na área da IoT. O conceito desta aplicação consiste em ter um nodo a fazer recolha de medições de temperatura e humidade periódicas (nodo 10 da Figura 7) e encaminhá-las para um outro nodo da rede.

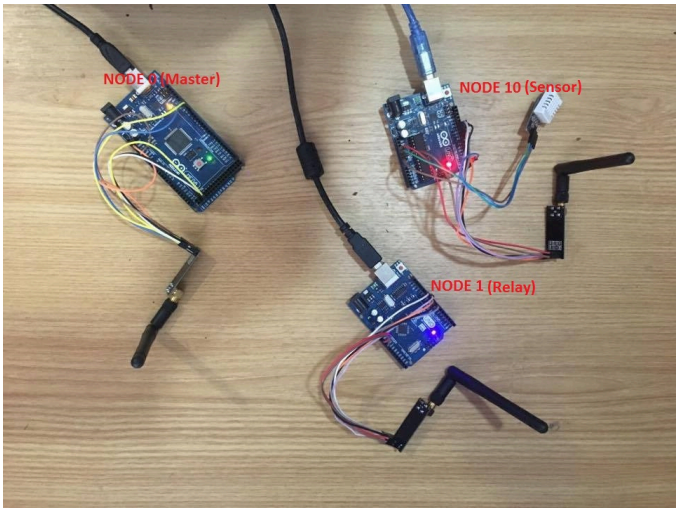

Figura 7. Topologia e dispositivos IoT usados nos testes.

Neste segundo teste, os dados são enviados de forma fiável ou não fiável mediante parametrização a nível da aplicação. Neste teste, é também forçado o cenário de operação *multi-hop* (na Figura 7, o nodo 10 é filho do nodo 1, que por sua vez é filho do nodo 0). As medições são recolhidas utilizando o sensor DTH22 [12], que se encontra interligado a um Arduino. Neste teste específico, as medições efetuadas que chegam ao seu destino (nodo 0 da Figura 7) são apresentadas no *output* da consola do Arduino, tal como descrito no teste anterior. Na Figura 7, apresenta-se o *hardware* utilizado nesta experiência demonstrativa.

Todas as etapas efetuadas no trabalho desenvolvido encontram-se disponíveis em [13].

V. CONCLUSÕES

Neste artigo apresentou-se uma extensão protocolar vocacionada para ambientes IoT. Tendo como ponto de partida algumas bibliotecas já existentes e de uso generalizado, o foco foi dotar estes ambientes IoT de soluções protocolares versáteis, capazes de suportar uma maior abrangência geográfica dos dispositivos IoT (e.g. comunicações *multi-hop*), bem como enriquecer os processos de comunicação que requeiram troca fiável de dados. Desta forma, a solução proposta permite diversificar e melhorar a operacionalidade das aplicações IoT a serem desenvolvidas nos mais variados cenários, em particular as que lidam com dados mais críticos, tais como aplicações na área da emergência médica e ambiental, entre muitas outras.

Após efetuar a descrição das bibliotecas de suporte à API desenvolvida, apresentou-se o protocolo proposto, descrevendo a sintaxe e a semântica dos PDUs, e das mensagens protocolares, bem como, os diagramas de transição de estado que regulam de forma clara o comportamento protocolar associado. Este último aspeto foi particularmente importante, dada a natureza *single-thread* das plataformas utilizadas, requerendo pois soluções eficazes para lidar com o possível envio e receção simultâneo de dados por parte dos dispositivos IoT. Por fim, alguns testes ilustrativos foram apresentados por forma a corroborar a eficácia da solução proposta.

Como trabalho futuro pretende-se aprofundar os testes efetuados à extensão protocolar e proceder-se ao desenvolvimento de novas aplicações demonstrativas das capacidades do protocolo proposto. De igual modo, serão também alvo de estudo futuro questões relacionadas com a segurança na troca de dados entre dispositivos IoT que utilizam a extensão protocolar apresentada.

AGRADECIMIENTOS

Este trabalho foi apoiado pelos projecto COMPETE: POCI-01-0145-FEDER-007043 e pela Fundação para a Ciência e Tecnologia no âmbito do projecto UID/CEC/00319/2013.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (Iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013, including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services and Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[2] N. Semiconductor, "nrf24l01+, Single chip 2.4GHz transceiver," https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Plus s_ Preliminary_Product_Specification_v1_0.pdf, 3 2008, acedido em 2017-06-10.

[3] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th Edition)*, 6th ed. Pearson, 2012.

[4] N. Semiconductor, "*Enhanced ShockBurst*^{TM} user guide," https://devzone.nordicsemi.com/documentation/nrf51/4.3.0/html/group_ _esb__users__guide.html, acedido em 2017-06-13.

[5] D. Sharma, S. Verma, and K. Sharma, "Network topologies in wireless sensor networks: A review," *International Journal of Electronics and Communication Technology*, vol. 4, pp. 93 – 97, 2013.

[6] D. Veilleux, "Intro to shockburst/enhanced shockburst," https://devzone. nordicsemi.com/blogs/783/intro- to- shockburstenhanced- shockburst/, acedido em 2017-06-14.

[7] V. Freitas, Apontamentos da disciplina de Fundamentos das Telecomunicações. Universidade do Minho, 2003.

[8] F. Leens, "An introduction to i2c and spi protocols," *IEEE Instrumentation and Measurement Magazine*, pp. 8 – 13, 2 2009.

[9] TMRh20, "Rf24," https://github.com/nRF24/RF24, acedido em 2017-02- 18.

[10] ——, "Rf24network," https://github.com/nRF24/RF24Network, acedido em 2017-02-18.

[11] ——, "Rf24mesh," https://github.com/nRF24/RF24Mesh, acedido em 2017-02-18.

[12] L. Aosong Electronics Co., "Digital-output relative humidity and temperature sensor/module dht22," https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf, acedido em 2017-06-15.

[13] "Wirlesssensornetwork",https://github.com/jms05/WirelessSensorNetwork, acedido em 2017-06-25.